Lab #6 – Drawable/Scaleable Shapes

Total Points: 50

Introduction

In the last lab you created a number of shapes that implemented the `Drawable` and `Scaleable` interfaces. You also implemented the `DrawableObjectList` that was a collection of `Drawable` shapes. In that lab you likely noticed a good bit of repeated code. In this lab you will use inheritance to implement a wider variety of shapes with less repetition of code. Specifically you will implement and test classes named `Line`, `Text`, `Rectangle`, `Square`, `Ellipse` and `Circle` that represent the corresponding objects.

Getting Started

1.Paste the Lab-6 link in the address bar of a browser window (after logging in to GitHub).

2.Accept the assignment.

3.GitHub will display a link, clicking that link will take you to the Lab-6 code repository.

4.Clone the Lab-6 repository to your computer by selecting the option "Clone a Git Repository and add it to this view" in the Git Repositories window in Eclipse. Note: In the cloning window, you should paste the web-link from the Lab-6 assignment repository on GitHub.com (click on Clone or Download and copy the link).

5.Once the Lab-6 project has been cloned, it will appear in the Git Repositories window. Import this in to the Package Explorer Window.

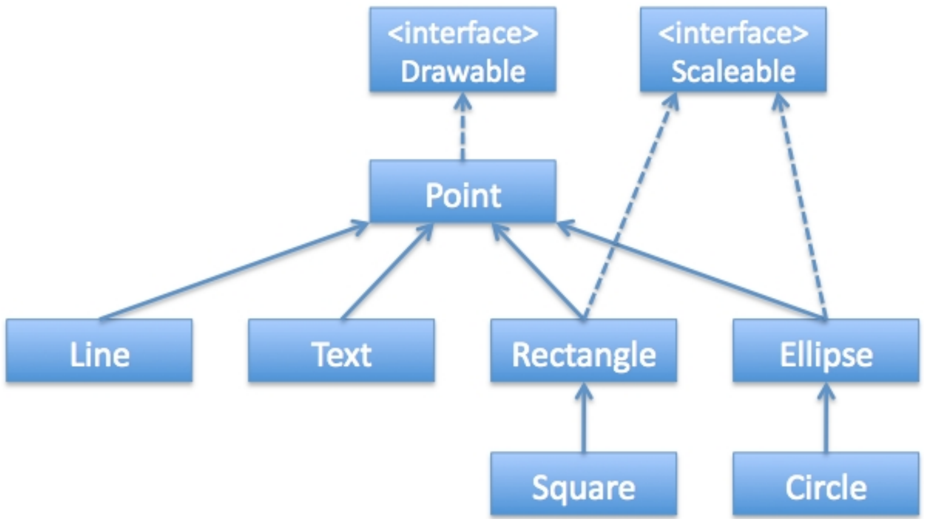
6.Expand the `src` folder of the Lab-6 project. It should contain a package named `shapes`.

Design

Provided with the lab is the class `Point` that implements the `Drawable` interface. A `Point` represents a singled colored pixel at a given (x,y) coordinate. The `Point` class provides implementations of all of the methods in the `Drawable` interface. It also contains two methods `move` and `translate` that change the position of the point in different ways (see the documentation in the `Point` class for more information). Each of the new classes that you create will be a sub-class of `Point` (or a sub-class of one of its sub-classes).

The `Point` class and all of its sub-classes form what is called an *inheritance hierarchy*. The inheritance hierarchy that you will be implementing can be represented visually as shown below. The `Drawable` and `Scaleable` interfaces are also shown. A solid arrow from one class to another indicates a sub-class / super-class relationship (e.g. `Line` is a sub-class of `Point`).

Dashed lines indicate that a class implements an interface (e.g. `Ellipse` implements the `Scaleable` interface).



The Assignment

Your assignment for this lab is to implement and test classes named `Line`, `Text`, `Rectangle`, `Square`, `Ellipse` and `Circle` that represent the corresponding objects. Once they are complete you must create another class that displays a picture or animation using at least one of each type of object.

The classes that you are creating must be implemented using the inheritance hierarchy described above in the Design section. Each class should inherit as much functionality as possible from its super-class. Each class should also override any inherited methods that need to behave differently in the sub-class. Every class must have a Java Doc comment at the top describing the purpose of the class and every method in every class must have a Java Doc comment describing its functionality. Each class must also have an associated JUnit test class that tests its functionality. Note that once a piece of functionality is tested in a super-class it does not need to be tested again in any sub-classes. Thus, for sub-classes only new and overridden methods need to be tested.

Submitting your solution

Turn in your solution by pushing your modifications to Lab-6 on GitHub as described in the **How-To** Document for the course (or refer back to lab # 1 instructions).